Commandes Git Avancées

Git est comme un couteau suisse pour les développeurs—polyvalent, puissant et parfois déroutant si vous ne savez pas quel outil utiliser. Aujourd'hui, nous allons explorer certaines des fonctionnalités et flux de travail les plus astucieux de Git : la sélection de modifications, la fusion avec style, le rebase pour une histoire plus propre, la suppression de ces fichiers volumineux que vous avez accidentellement commis, et l'annulation d'un commit lorsque vous réalisez que vous avez déraillé. Décortiquons cela.

Sélection de modifications : Prendre exactement ce dont vous avez besoin Imaginez que vous avez une branche de fonctionnalité avec une douzaine de commits, mais il y a un commit brillant que vous voulez extraire et appliquer à votre branche principale—sans apporter le reste. C'est là qu'intervient `git cherry-pick`.

C'est super simple : trouvez le hash du commit (vous pouvez le récupérer avec `git log`), passez à la branche où vous le voulez, et exécutez :

```
git cherry-pick <commit-hash>
```

Boum, ce commit fait maintenant partie de votre branche actuelle. Si un conflit survient, Git s'arrêtera et vous laissera le résoudre, comme une fusion. Une fois que vous êtes satisfait, validez les modifications, et c'est bon.

Je l'utilise tout le temps lorsqu'une correction de bug se faufile dans une branche de fonctionnalité désordonnée, et que j'en ai besoin sur `main` le plus vite possible. Mais soyez prudent—la sélection de modifications duplique le commit, donc il obtient un nouveau hash. Ne vous attendez pas à ce qu'il fonctionne bien si vous fusionnez la branche originale plus tard sans un peu de nettoyage.

Options de fusion : Plus qu'une simple fusion La fusion est le pain et le beurre de Git, mais saviez-vous qu'elle vient avec des saveurs ? La fusion par défaut `git merge` fait un "fast-forward"si possible (redressant l'historique) ou crée un commit de fusion si les branches ont divergé. Mais vous avez des options :

- `--no-ff` **(Pas de Fast-Forward)** : Force un commit de fusion même si un fast-forward est possible. J'adore ça pour garder un enregistrement clair de quand une branche de fonctionnalité a atterri sur `main`. Exécutez-le comme ceci :

  ```
  git merge --no-ff feature-branch
  ```

- `--squash` : Rassemble toutes les modifications de la branche en un seul commit sur votre branche actuelle. Pas de commit de fusion, juste un seul paquet bien rangé. Parfait pour aplatir une branche désordonnée en quelque chose de présentable :

```
git merge --squash feature-branch
```

Après cela, vous devrez valider manuellement pour sceller l'affaire.

Chacune a sa place. Je penche vers `--no-ff` pour les branches à long terme et `--squash` lorsque j'ai une branche pleine de commits "WIP"que je préférerais oublier.

Rebase : Réécrire l'histoire comme un pro Si les fusions vous semblent trop encombrées, `git rebase` pourrait être votre truc. Il prend vos commits et les rejoue sur une autre branche, vous donnant un historique linéaire qui semble que vous avez tout planifié parfaitement dès le départ.

Passez à votre branche de fonctionnalité et exécutez :

```
git rebase main
```

Git enlèvera vos commits, mettra à jour la branche pour correspondre à `main`, et remettra vos modifications par-dessus. Si des conflits apparaissent, résolvez-les, puis `git rebase --continue` jusqu'à ce que ce soit terminé.

Le côté positif ? Une chronologie impeccable. Le côté négatif ? Si vous avez déjà poussé cette branche et que d'autres y travaillent, le rebase réécrit l'histoire—bienvenue les emails en colère de vos collègues. Je m'en tiens au rebase pour les branches locales ou les projets solo. Pour les choses partagées, la fusion est plus sûre.

Suppression de grands fichiers de l'historique : Oups, cette vidéo de 2 Go Nous y sommes tous passés : vous avez accidentellement validé un fichier massif, l'avez poussé, et maintenant votre repo est gonflé. Git n'oublie pas facilement, mais vous pouvez effacer ce fichier de l'historique avec un peu d'effort.

L'outil de prédilection ici est `git filter-branch` ou le plus récent `git filter-repo` (je recommande ce dernier —il est plus rapide et moins sujet aux erreurs). Disons que vous avez validé `bigfile.zip` et que vous en avez besoin : 1. Installez `git-filter-repo` (consultez sa documentation pour la configuration). 2. Exécutez : `git filter-repo --path bigfile.zip --invert-paths` Cela supprime `bigfile.zip` de chaque commit dans l'historique. 3. Forcez la poussée de l'historique réécrit : `git push --force`

Attention : cela réécrit l'historique, donc coordonnez-vous avec votre équipe. Et s'il est dans une demande de tirage quelque part, vous devrez peut-être nettoyer les références aussi. Une fois qu'il est parti, votre repo s'allégera après une collecte des déchets (`git gc`).

Annulation de la validation : Rembobiner l'horloge Vous avez fait un commit et l'avez instantanément regretté ? Git vous couvre. Il y a quelques façons de l'annuler, selon la distance que vous avez parcourue :

- **Si vous n'avez pas encore poussé** : Utilisez `git reset`. Pour annuler le dernier commit mais garder les modifications dans votre répertoire de travail :

```
git reset HEAD^ --soft
```

Vous voulez vous débarrasser des modifications entièrement ?

```
git reset HEAD^ --hard
```

- **Si vous avez déjà poussé** : Vous devrez réécrire l'historique. Réinitialisez localement avec `git reset HEAD^`, puis forcez la poussée :

  ```
  git push --force
  ```

  Encore une fois, cela perturbe l'historique partagé, alors soyez prudent.

J'ai sauvé avec `git reset --soft` plus de fois que je ne peux le compter—parfait lorsque je valide trop tôt et que j'ai besoin de retoucher quelque chose.

Conclusion La flexibilité de Git est ce qui le rend si puissant, mais il est facile de s'embrouiller si vous ne connaissez pas vos options. Sélectionnez pour une précision chirurgicale, ajustez les fusions pour qu'elles s'adaptent à votre flux de travail, rebasez pour un historique poli, et ne paniquez pas lorsque vous devez effacer une erreur—que ce soit un gros fichier ou un commit hâtif. Pratiquez ces techniques sur un repo de test si vous êtes nerveux, et bientôt elles vous sembleront naturelles.

Quel est votre astuce Git préférée ? Faites-le moi savoir—je suis toujours partant pour apprendre quelque chose de nouveau !